WebAssemblyにおけるメモリ使用量削減のための稼働実績に基づく再起動の閾値の決定

中川 翔太[1]　串田 高幸[1]

概要：Web ブラウザ上で計算を高速に行うために登場した WebAssembly は，ブラウザ外でも利用され始めている．WebAssembly では，インスタンスの終了まで 1 度拡張した最大量のメモリを確保し続けることが課題である．本研究では，インスタンスのメモリ使用量に閾値を設け，閾値を超えたインスタンスの再起動する．これによって，インスタンスによる実行環境のメモリ使用量を削減する．閾値は，インスタンスのメモリ使用量の分布が正規分布になると仮定のもと，再起動処理を行わない場合のインスタンスのメモリ使用量のデータを標本として区間推定を行い決定する．実験では，再起動処理を行わなかった場合と，提案手法を用いて再起動を行った場合とで，メモリ使用量の差分について比較した．計測は同一のインスタンスを 50 個づつ 6 分間動作させて実施した．実験対象は，不定期にメモリ使用量が増加し，最大で約 8000[KiB] まで増加するインスタンスを用いた．実験の結果，平均メモリ使用量は，計測のみ実施した場合は 6541[KiB] であり，提案手法を適用した場合は 6115[KiB] であった．したがって，提案手法の適用により，監視のみの場合と比較して 6.52%メモリ使用量の削減がされた．

1. はじめに

背景

WebAssembly は，Web ブラウザ上で動作する低レベルコードであり，コンパクトな表現，効率的な検証とコンパイル，オーバーヘッドのない安全な実行を提供する [1]．WebAssembly は，ポータビリティとセキュリティに優れるという特徴から，Web ブラウザ外でも利用され始めている．例えば，ブラウザ外のシステムとのインタフェースの標準化を図る WebAssembly System Interface (WASI)[*1] の策定が進められている．活用事例として，WebAssembly for Proxies[*2] では，L4/L7 プロキシのプラグインとして WebAssembly を利用している．また，Krustlet[*3] では，WebAssembly のインスタンスを Kubernetes の Pod として，直接実行させている．Fastly 社のプロダクトの 1 つである Compute@Edge[*4]では，ネットワークのエッジでのリクエストの処理に WebAssembly を活用している．

WebAssembly では，1 つの WebAssembly バイナリ (.wasm) あるいは WebAssembly テキストフォーマット (.wat) を 1 つのモジュールとして扱う．WebAssembly では，モジュールをランタイム上に展開したものをインスタンスとして実行する．

WebAssembly は線形メモリモデルを採用している．線形メモリモデルでは，メモリはインスタンス及びランタイムから 1 つの連続したアドレス空間として見える [2]．各モジュールはメモリ空間を 1 つ定義できる．モジュールからロードされたインスタンスは，起動時にモジュールにある定義に基づいてメモリを確保する．また，必要に応じてメモリをページサイズ (64KB) 単位で拡張できる [1]．インスタンスは確保したメモリ空間を用いて処理を行い，不足した分だけメモリ空間の拡張をランタイムに要求する．しかし，拡張されたメモリを部分的に解放，あるいは縮小する機能はサポートされていない．一度確保されたメモリ空間はインスタンスの終了まで解放されず，そのメモリ空間の大きさは同時に利用したメモリ使用量の最大量になる．したがって，一度メモリ空間が拡大されてから，次にメモリ空間が拡大されるまでのメモリ使用量は一定になる．

課題

WebAssembly インスタンスが利用するメモリは，一度拡大されると縮小することができない．したがって，一度大きな容量のメモリが確保されるとその後の利用実態にかかわらず，その WebAssembly インスタンスが使用するメ

[1] 東京工科大学大学院バイオ・情報メディア研究科コンピュータサイエンス専攻
〒 192–0982 東京都八王子市片倉町 1404–1

[*1] https://wasi.dev/
[*2] https://github.com/proxy-wasm
[*3] https://krustlet.dev/
[*4] https://docs.fastly.com/products/compute-at-edge

モリは解放されない．したがって，インスタンスのメモリ使用量を削減するためには，インスタンスの再起動によって 1 度メモリを解放する必要がある。しかし，頻繁なインスタンスの再起動はインスタンスを長期間稼働させるという目的に反する．そのため，どのインスタンスをいつ再起動するかが課題となる．

各章の概要

2 章では，関連研究を紹介する．3 章では，提案方式とそのユースケース・シナリオについて説明する．4 章では，提案方式に基づいたソフトウェアの実装と，その実験方法について説明をする．5 章では，提案方式とそのソフトウェアの評価方法及び，得られるデータの分析手法について説明をする．6 章では，本研究の提案方式について議論をする．7 章では，本研究のまとめを述べる．

2. 関連研究

Lehmann らは，WebAssembly を動的解析するための汎用フレームワーク Wasabi を提案している [3]．事前に JavaScript で実装された解析するためのコードをバイナリに挿入をして実行することにより，実行時のパフォーマンスのボトルネック，エラーやセキュリティギャップの検出が可能である．動的解析はプロダクション環境のアプリケーションに対しては利用するためのものではない．そのため，安定して長期間 WebAssembly アプリケーションを運用をするためには Wasabi のような動的解析機構の他に，恒常的な監視のための機構を検討する必要がある．

Mäkitalo らは，ブラウザ外でクロスプラットフォームのモジュラーアプリケーションを構築するために WebAssembly を利用することを検討している [4]．WebAssembly の実行時のダイナミックリンクの実装と測定を行った結果，モジュール化によってアプリケーションの起動時間が約 98%短縮されている．また，ダイナミックリンクによる実行時間への影響はごくわずかであり，モジュールから 10 分の 1 だけ機能をロードする度に実行時間が約 90%短縮されている．今後の方向性では，モジュールのプリロードによる起動時間のオーバーヘッドの最小化や，使用されていないインスタンスを検出してアンロードすることによるシステムリソースの管理，複数アプリケーション間におけるモジュールのキャッシュの共有を挙げている．使用されていないインスタンスの検出だけでなく，メモリ使用量が〜インスタンスについても検出し，再起動をすることによって，より効率のよいシステムのメモリ使用量についての管理が可能になる．

Elliott らは，リソースに制約のある IoT やフォグデバイスで WebAssembly を実行する OS である Wasmachine を紹介している [5]．この研究では，WebAssembly のネイティブコンパイルを行い，カーネルモードで実行することで Linux と比較して最大 11%高速化している．また，WebAssembly のサンドボックス機能のほか，Rust 言語によるメモリ安全性の担保によって高いセキュリティ性能を維持している．現時点ではリソースプロビジョニング機能が欠けていることに言及しており，マルチタスクを実行する際のプロセスの実行時間や最大メモリ量の制限の制限が実行できない．また，全てのプロセスがインメモリに収まることを前提としているため，インメモリに収まらない場合を考慮したメモリに関するスケジュール機構を検討する必要がある．

3. 提案方式

本研究では，WebAssembly インスタンスのメモリ使用量に対して閾値を設けて，閾値を超えたメモリ使用量のインスタンスの再起動によって，実行環境におけるメモリ使用量を削減することを提案する．閾値には，インスタンスのメモリ使用量の分布が正規分布になるという仮定のもとで区間推定を行い，区間における最大値を用いる．

基礎実験

基礎実験では，WebAssembly アプリケーションとネイティブアプリケーションをそれぞれ実行し，実行時の物理メモリの使用量 (RSS) を取得し比較する．この基礎実験によって，WebAssembly のインスタンスが利用するメモリは，一度拡大された後インスタンスの終了まで解放されないことを確認する．

測定対象として，Rust 言語を用いて毎秒ランダムなサイズ (16[KB]–16384[KB]) のメモリの確保と解放を繰り返すアプリケーションを作成した．そして，同一のソースコードをコンパイルし，Webassembly アプリケーションとネイティブアプリケーションを生成した．

ps コマンドを毎秒繰り返し実行することで，Webassembly アプリケーションとネイティブアプリケーションの実行時の RSS を収集した．図 1 は，ネイティブアプリケーションの RSS の推移を示したグラフである．起動直後に 30MB 程度読み込まれたあと，RSS が増減を繰り返していることがわかる．図 2 は，WebAssembly アプリケーションの RSS の推移を示したグラフである．起動直後に 80 [MB] 程度読み込まれたあと，20 秒後に約 90[MB] に上昇し，それ以降は一度も RSS 値が減少していないことがわかる．

このことから，確かに WebAssembly のインスタンスは，一度確保したメモリ空間をそのインスタンスの終了まで解放しないことが確認できる．

提案方式

本研究の提案では，WebAssembly インスタンスの再起

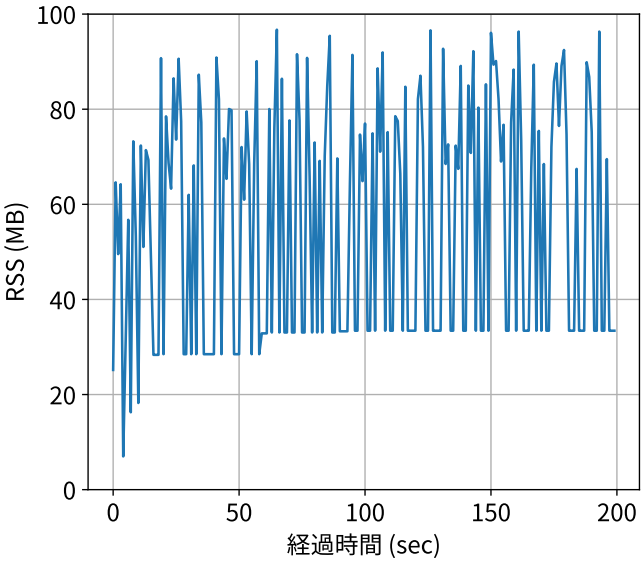

図 1　ネイティブアプリケーションとして実行

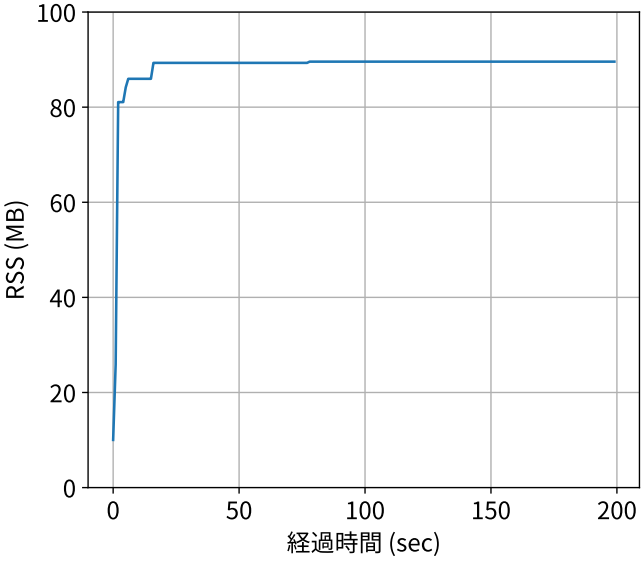

図 2　WebAssembly アプリケーションとして実行

動に利用するメモリ使用量の閾値を決定する．

本研究で用いる WebAssembly アプリケーションでは，インスタンスが確保するメモリの容量が外部からに入力に依存する．そのため，外部からの入力の大きさと量が同じであれば，処理に必要なメモリ使用量は同じになる．インスタンスが複数ある場合に，その全てのインスタンスへの入力は，通常同一にはならないため，複数インスタンスのメモリ使用量は，全てが同一にはならない．また，インスタンスへの入力に用いられるファイルのサイズには上限があり，そのファイルのサイズの上限と WebAssembly インスタンスおよびランタイムの合計より大きいメモリ使用量にはならない．

さらに，ユースケース・シナリオで示すようなオンラインストレージサービスの場合，アップロードされるファイルは画像ファイルがボリュームゾーンとなる想定をしている．また，画像ファイルよりもファイルサイズの小さいテキストファイルや，画像ファイルよりもファイルサイズの大きい映像ファイルは，画像ファイルよりもアップロードされる総数は少ない．結果的に正規分布に近しい山なりのファイルサイズの分布になると想定している．本研究では，想定される入力で与えられるデータの大きさの分布は，簡単のため正規分布に従うものと仮定する．したがって，各インスタンスについて，メモリ使用量の安定値の分布も同様に正規分布になる．

推定アルゴリズム

本項では，再起動するインスタンスの選択に用いる閾値の決定手法を説明する．

はじめに，同一モジュールをロードして実行される複数のインスタンスのメモリの容量の中央値 a の集合を母集団 P とおく．これは，$P = \{a_1, a_2, \ldots, a_n\}$ と表記でき，n は，インスタンスの総数である．しかし，図 2 における 0–20 秒のような，インスタンスの起動直後のメモリ使用量の値を含めた場合，メモリ使用量が増加している最中の値が反映される．そのため，本来標本として利用したい安定値よりも，中央値は小さくなる．したがって，各インスタンスについて求めるメモリ使用量の中央値の算出では，メモリ使用量の増加傾向の停止以降の，図 2 の場合では 20 秒以降の，値を利用する．

ここで，P は正規分布に従う前提のもと，P の母平均 μ を推定する．推定値は，μ をインスタンスのメモリの容量の必要最小量として，区間推定を行う．本研究の提案では，インスタンスが生成されてから現在までに得られているメモリ使用量の中から m 個を取り出した a の集合を標本 $x = \{a_1, a_2, \ldots, a_m\}$ として利用する．母平均 μ の信頼区間を求める際は通常，母分散 σ^2 利用する．しかし，σ^2 は (1) 式で求める必要があるが，未知数である母平均 μ に依存しているため，利用できない [6].

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2 \tag{1}$$

したがって，σ^2 を利用する代わりに，推定する時点で判明しているメモリの容量のデータから求められ，期待値が σ^2 により近くなる分散である，不偏分散 u^2 を利用する．不偏分散 u^2 は，(2) 式で求めることができる [6].

$$u_2 = \frac{1}{m-1}\sum_{i=1}^{m}(x_i - \overline{x})^2 \tag{2}$$

区間推定に u^2 を利用する場合の求まる分布は正規分布ではなく t 分布になる．t 分布は標本数の増加につれ，正規分布に近似していく特徴がある．そのため，P が確かに正規分布に従っている場合，メモリ使用量のデータの数が多くなるにつれ推定の精度が向上する．また，区間推定に用いる標本平均 $\overline{x}$ は，(3) 式で求まる [6].

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{3}$$

更に，t 分布における区間推定では，統計量である t 値を予め計算しておく必要がある．自由度が $m-1$ で，信頼水準を $1-\alpha$ とおいたときの上側確率が $\alpha/2$ となる t 分布の場合，その t 値は $t_{\alpha/2}(n-1)$ である．求めた u_2 と $\overline{x}$ を利用して，(4) によって母分散 σ^2 が未知である場合の母平均 μ の信頼区間を求めることができる [6]．

$$\overline{x} - t_{\alpha/2}(n-1)\sqrt{\frac{u^2}{n}} \leq \mu \leq \overline{x} + t_{\alpha/2}(n-1)\sqrt{\frac{u^2}{n}} \tag{4}$$

本研究の提案手法では，推定された区間の最大値である $\overline{x} + t_{\alpha/2}(n-1)\sqrt{\frac{u^2}{n}}$ を閾値として利用する．この閾値を超えたメモリの容量を確保している WebAssembly のインスタンスを再起動することで，WebAssembly インスタンスのメモリ使用量を削減する．

ユースケース・シナリオ

アプリケーションとして，24 時間 365 日稼働しているオンラインストレージサービスを想定する．Web ブラウザベースでのファイルのアップロード・ダウンロード，ファイルの一覧とメタデータの表示をサポートする．取り扱うファイルは，数 KB のテキストファイルから数百 MB の動画ファイルまで，大小は様々なサイズが存在する．

サーバー側は，図 3 の構成であり，HTTP リクエストを受け取って以下の処理をする．

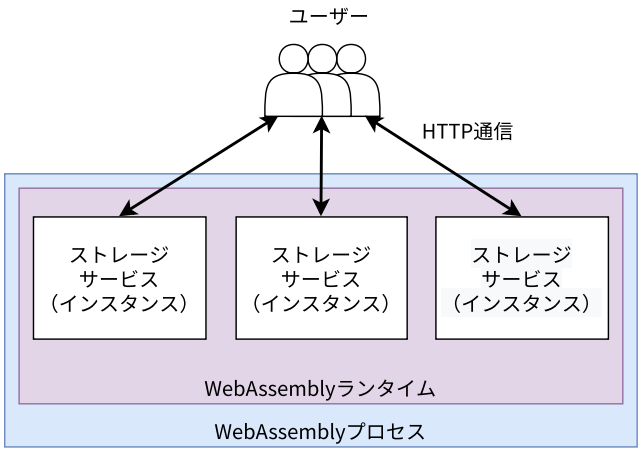

図 3　アプリケーションインスタンスの配置

`GET /path`　`path` に対応するファイルのメタデータを Volume から取得する．

`POST /path`　リスエストボディの書き込まれた，`path` に対応するファイルを Volume に作成する．

`DELETE /path`　`path` に対応するファイルを Volume から削除する．

本研究の提案機構を上記のユースケースに適用する場合，初めにオンラインストレージサービスの動作環境に配置しストレージサービスのインスタンスを監視する．そして，取り扱うファイルの平均的なサイズよりも大きいファイルの読み込みによって，メモリ使用量が増加したインスタンス検出し，再起動する．インスタンスの再起動にしたがって，インスタンスの使用するメモリも 1 度解放される．これによって，インスタンスのメモリ使用量が削減される．

4. 実装と実験方法

実装

本研究における実装では，機能を以下の 3 つに大分できる．

- 各インスタンスの起動と削除をする機能．
- 各インスタンスのメモリの容量を一定間隔で取得する機能．
- 各インスタンスのメモリ統計をもとに再起動のための閾値を決定する機能．

図 4 に実装の全体の構造を示す．本研究では，メモリ使用量の取得の簡単のため，WebAssembly のインスタンス 1 つを 1 つのサブプロセスとして動作させている．Worker プロセスでは，ハンドラによって WebAssembly のサブプロセスの起動と停止をする．WebAssembly のランタイムには Wasmtime を利用し，1 つのサブプロセスに 1 つの WebAssembly インスタンスを動作させる．また，メモリ取得機構では，WebAssembly のプロセスのプロセス番号 (PID) を用いて procfs からメモリ使用量 (USS) 取得する．取得は Julian. M の Practical Monitoring を参考に 10 秒毎に行う [7]．取得したメモリ使用量の情報は，タイムスタンプを付与したうえで CSV ファイルへ書き出す．Estimator プロセスでは，CSV へ書き出されたメモリ使用量のデータを利用して，閾値決定機構にて，提案手法を用いて再起動のための閾値を決定する．Worker プロセスのため設定ファイルへその閾値を書き込むことによって，それ以降 WebAssembly アプリケーションを実施した際にその閾値に則った再起動がハンドラによって行われる．

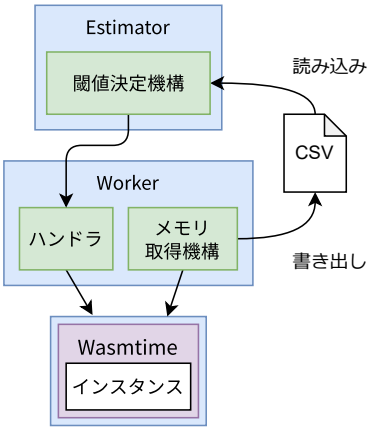

図 4　実装の概要図

実験環境

実験環境では, 図 5 に示すようにハイパーバイザ (VMware ESXi) 上の仮想マシンに Ubuntu 20.04 をインストールして利用する．仮想マシン上に図 4 の Worker アプリケーションと，Estimator アプリケーションをデプロイする．そして，Worker アプリケーションを通して WebAssembly のインスタンスを動作及びメモリ使用量の収集をする．

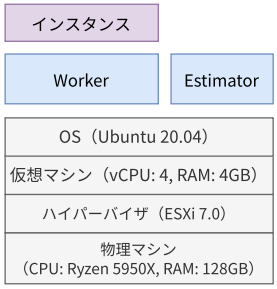

図 5　実験環境の外観

5.　評価と分析

提案手法を用いた閾値の決定による再起動をおこなう場合と，再起動を行わない場合によるインスタンスのメモリ使用量の差分を比較し評価する．

評価に用いる WebAssembly アプリケーションは，Rust 言語を用いて作成した．この WebAssembly アプリケーションでは，10 秒に 1 回ランダムに選択したファイルを読み込み，読み込んだデータのハッシュ値を計算する．したがって，WebAssembly アプリケーションのメモリ使用量は，読み込むファイルサイズに依存する．WebAssembly アプリケーションが起動してから，今までに読み込んだファイルよりも大きいファイルが読み込まれた際に，その差分だけ WebAssembly アプリケーションのメモリ使用量は増加する．WebAssembly アプリケーションが読み込むファイルは，numpy を用いて作成するファイルサイズの分布が正規分布になるよう計算し，dd コマンドを用いてそのサイズのファイルを作成した．図 6 に示す分布になるように，合計 1024 個のファイルを作成した．

はじめに，WebAssembly アプリケーションを動作させて，メモリ使用量の取得のみ行った．並列で 50 個同一のアプリケーションをプロセスとして起動し，10 分間メモリ使用量を 10 秒おきに取得した．メモリ使用量の取得では，procfs を利用し，Unique Set Size (USS) を計算して求めた．その結果を図 7 に示す．起動からおよそ 10 秒後のメモリ使用量は 2948[KiB] から 5384[KiB] であった．その後，ファイルの読み込みによって各 WebAssembly アプリケーションのメモリ使用量は増加し，最大で 8184[KiB] であった．また，各プロセスのメモリ使用量の中央値のヒストグラムは図 8 になった．プロットされた点は，その区間における平均値である．分布が最も集中した区間は，$[6524, 6690)$ の 10 個であった．

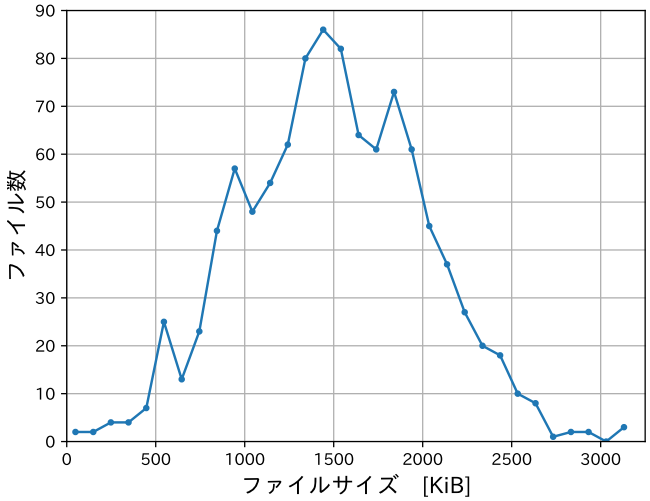

図 6　dd コマンドを用いて生成したファイル群のサイズの分布

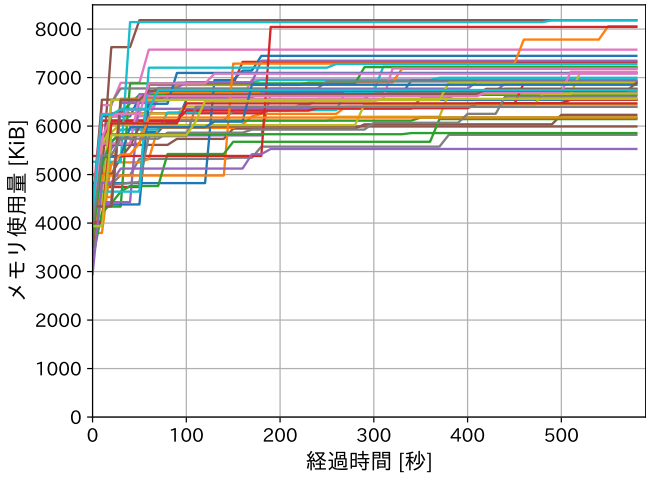

図 7　監視のみ行った場合のメモリ使用量の推移

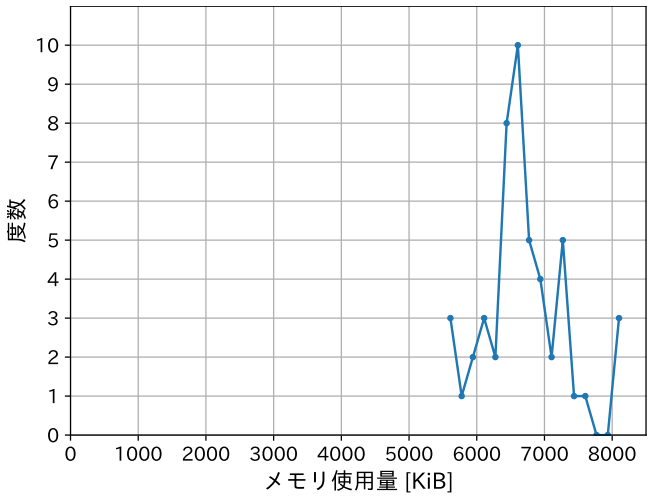

図 8　監視のみ行った場合のメモリ使用量の中央値の分布

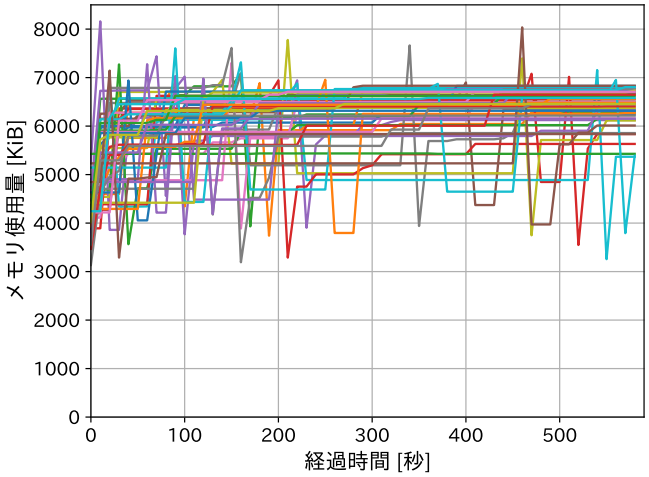

図 9　提案手法を用いた場合のメモリ使用量の推移

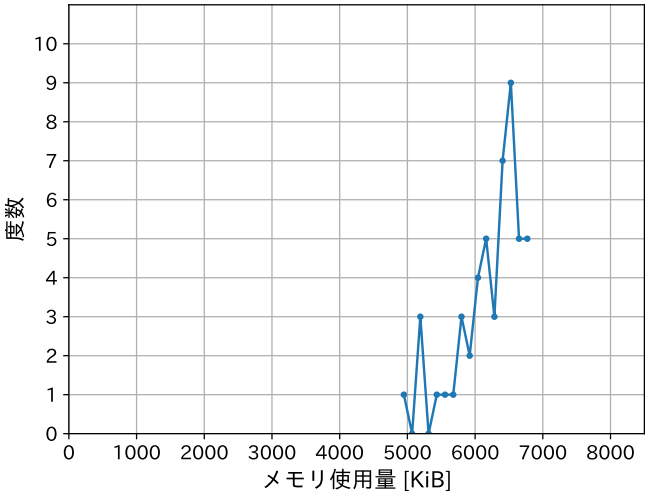

図 10　提案手法を用いた場合のメモリ使用量の中央値の分布

次に，提案手法を適用した場合のメモリ使用量を取得した．区間推定には，図 7 の値を標本として利用した．また，本研究では上側 2.5%点を閾値として利用した．計算では scipy を利用し，閾値は 6859[KiB] である．提案手法を適用し，閾値として 6859[KiB] を適用した場合の，WebAssembly アプリケーションのプロセス 50 個のメモリ使用量の推移を図 9 に示す．メモリ使用量が閾値として設定された 6859[KiB] を超えたプロセスが再起動され，メモリ使用量が減少している．そして，提案手法を適用した場合の各プロセスのメモリ使用量の中央値のヒストグラムは図 10 のようになった．分布が最も集中した区間は $[6467, 6589)$ の 9 個であった．

図 9 の結果における再起動の発生する頻度は，平均で 789 秒に 1 回であった．本研究の評価における実験では，600 秒間 10 秒毎にファイルを読み込んでいる．すなわち，1 つの WebAssembly アプリケーションのプロセスにつき合計 60 回ファイルを選択して読み込んでいる．WebAssembly アプリケーションのメモリ使用量が増加する機会は，起動時を除いてファイルの読み込み時のみである．したがって，評価実験におけるプロセスのメモリ使用量が閾値を越える確率は，1.2%である．

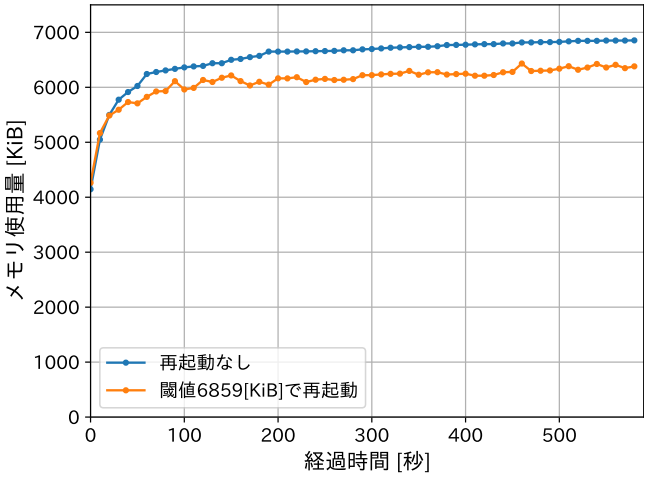

図 11　提案手法を適用した場合と監視のみ行った場合の平均の比較

また，図 11 では，単にメモリ使用量を取得した場合の 50 個のプロセスの平均メモリ使用量の推移と，提案手法を適用し 6859[KiB] で再起動をした場合のメモリ使用量の平均の推移についてプロットし，比較した．監視のみ行った場合の平均メモリ使用量は 6541[KiB]，提案手法を用いた場合の平均メモリ使用量は 6114[KiB] である．したがって，監視のみの場合と比較した場合の提案手法によるメモリ使用量の削減率は 6.52%でる．

6.　議論

再起動する WebAssembly インスタンスの判定に用いる閾値の決定プロセスで，1 つの WebAssembly インスタンスのメモリ使用量の代表値として中央値を利用してる．しかし，WebAssembly インスタンスのメモリの容量が緩やかに増加し続けた場合に，中央値は追従して増加し続ける．この場合，WebAssembly インスタンスの稼働時間に比例して求まる閾値も増加し続けるため，再起動が実施されない．

また，インスタンスのメモリ使用量の推移とその傾向は WebAssembly アプリケーション毎に異なる．本研究の前提条件である，WebAssembly インスタンスのメモリ使用量の分布が正規分布になるという制約は，適用可能な WebAssembly アプリケーションを限定してしまう．したがって，何らかのメモリ使用量の分布になるという前提条件を排した閾値の決定手法が必要である．

今後の作業では，WebAssembly アプリケーションを予め動作させ，計測することによってメモリ使用量の分布を作成する．そして，作成した分布をもとに再起動のための閾値を決定する．

7.　おわりに

WebAssembly のインスタンスが使用するメモリ空間は，

拡張可能であるが縮小はできない．そのためインスタンスの終了まで 1 度拡張した最大量のメモリを確保し続けることを課題とした．本研究では，インスタンスのメモリ使用量に閾値を設け，閾値を超えたインスタンスの再起動させ，インスタンスによる実行環境のメモリ使用量を削減する提案をした．閾値は，インスタンスのメモリ使用量の分布が正規分布になると仮定のもと，再起動処理を行わない場合のインスタンスのメモリ使用量のデータを標本として区間推定を用いて決定した．実験では，再起動処理を行わなかった場合と，提案手法を用いて再起動を行った場合とで，メモリ使用量の差分について比較した．計測は同一のインスタンスを 50 個づつ 6 分間動作させて実施した．実験対象には，不定期にメモリ使用量が増加し，最大で約 8000[KiB] まで増加するインスタンスを用いた．実験の結果，平均メモリ使用量は，計測のみ実施した場合は 6541[KiB] であり，提案手法を適用した場合は 6115[KiB] であった．したがって，計測のみ実施した場合と比較して，提案手法を適用した場合は，6.52%メモリ使用量の削減された．

参考文献

[1] Haas, A., Rossberg, A., Schuff, D. L., Titzer, B. L., Holman, M., Gohman, D., Wagner, L., Zakai, A. and Bastien, J.: Bringing the web up to speed with WebAssembly, *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 185–200 (2017).

[2] González, A., Latorre, F. and Magklis, G.: *Processor Microarchitecture: An Implementation Perspective*, Synthesis lectures on computer architecture, Morgan & Claypool Publishers (2010).

[3] Lehmann, D. and Pradel, M.: Wasabi: A framework for dynamically analyzing webassembly, *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 1045–1058 (2019).

[4] Mäkitalo, N., Bankowski, V., Daubaris, P., Mikkola, R., Beletski, O. and Mikkonen, T.: Bringing WebAssembly up to Speed with Dynamic Linking, *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, SAC '21, New York, NY, USA, Association for Computing Machinery, p. 1727–1735 (online), DOI: 10.1145/3412841.3442045 (2021).

[5] Wen, E. and Weber, G.: Wasmachine: Bring IoT up to Speed with A WebAssembly OS, *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pp. 1–4 (online), DOI: 10.1109/PerComWorkshops48775.2020.9156135 (2020).

[6] Mood, A. M., Graybill, F. A. and Boes, D. C.: *Introduction to the Theory of Statistics* (1974).

[7] Julian, M.: *Modern Monitoring* (2017).